Exploiting Tunability of Erasure Coding for Low-Interference Repair

Paper ID: 27

1 Summary of Implementations

We implement a prototype of ChameleonEC from scratch with around 3,000 lines of code (LoC) in C++. We employ `Jerasure Library v2.0` coupled with `GF-complete` to enable the encoding and decoding functionalities over `Galois Field` arithmetic. We also employ the interfaces of `Redis` to realize the data transmission and storage across nodes. We elaborate on the dependent software and testing tools as follows.

Dependent software: The implementation of ChameleonEC relies on a suite of software, including `Jerasure Library v2.0`, `GF-complete`, `Redis`, and `nethogs`, which can be reached via the following URLs.

```
#!/bin/bash
$ wget https://github.com/intel/intel
 -cmt-cat
$ git clone https://github.com/ceph/
gf-complete.git
$ git clone https://github.com/
tsuraan/Jerasure.git
$ wget https://github.com/raboof/
nethogs/archive/v0.8.5.tar.gz
```

Testing tools: we employ `YCSB`, `IBM Object Store`, `Memcached(Twitter)` (a trace released by Twitter), and `Facebook ETC workloads` in our evaluation. They can be reached via the following URLs.

```
#!/bin/bash
$ wget https://github.com/
brianfrankcooper/YCSB/releases/
download/0.17.0/ycsb-0.17.0.tar.gz
$ http://iotta.snia.org/traces/
key-value/28652
$ http://iotta.snia.org/traces
/key-value/36305
$ https://dl.acm.org/doi/pdf/
10.1145/2254756.2254766
```


```
<setting>
<attribute><name>erasure.code.type</name><value>RS</value></attribute>
<attribute><name>erasure.code.k</name><value>10</value></attribute>
<attribute><name>erasure.code.n</name><value>14</value></attribute>
<attribute><name>node.fail.cnt</name><value>1</value></attribute>
<attribute><name>lrc.code.l</name><value>0</value></attribute>
<attribute><name>period.len</name><value>20</value></attribute>
<attribute><name>encode.matrix.file</name><value>/root/ElasticRepair/conf/rsEncMat_10_14</value></attribute>
<attribute><name>packet.size</name><value>1048576</value></attribute>
<attribute><name>packet.count</name><value>64</value></attribute>
<attribute><name>single.stripe.alg</name><value>path</value></attribute>
<attribute><name>schedule.method</name><value>online</value></attribute>
<attribute><name>coordinator.address</name><value>172.31.0.201</value></attribute>
<attribute><name>file.system.type</name><value>standalone</value></attribute>
<attribute><name>meta.stripe.dir</name><value>/root/repairboost-code/meta/standalone-meta</value></attribute>
<attribute><name>file.system.type</name><value>standalone</value></attribute>
<attribute><name>block.directory</name><value>/root/repairboost-code/meta/standalone-blocks</value></attribute>
<attribute><name>helpers.address</name>
<value>default/172.31.0.202</value>
<value>default/172.31.0.203</value>
<value>default/172.31.0.204</value>
<value>default/172.31.0.205</value>
<value>default/172.31.0.206</value>
<value>default/172.31.0.207</value>
<value>default/172.31.0.208</value>
<value>default/172.31.0.209</value>
<value>default/172.31.0.210</value>
<value>default/172.31.0.211</value>
<value>default/172.31.0.212</value>
<value>default/172.31.0.213</value>
<value>default/172.31.0.214</value>
<value>default/172.31.0.215</value>
<value>default/172.31.0.216</value>
<value>default/172.31.0.217</value>
<value>default/172.31.0.218</value>
<value>default/172.31.0.219</value>
<value>default/172.31.0.220</value>
</attribute>
<attribute><name>local.ip.address</name><value>172.31.0.201</value></attribute>
</setting>
```


Figure 1: Configurations of ChameleonEC.

2 Evaluation Details

Hardware configurations: We assess the performance of ChameleonEC on Amazon EC2. We allocate 20 virtual machine instances with the type of `m5.xlarge` in the region of North Virginia. Each instance runs Ubuntu 16.04.7 LTS and owns four vCPU with 3.1 GHz Intel Xeon Platinum processor, 16 GB RAM, and 100 GB Elastic Block Store (EBS) volumes. The network bandwidth between any two instances is around 5 Gb/s (measured via `iperf`).

We employ the tool `wondershaper` to throttle the network bandwidth, which is set to 1 Gb/s by default.

```
# !/bin/bash
$ wondershaper -a ens5 -d 1048576 -u 1048576
```

We perform extensive testbed experiments, including (i) the experiments to understand the properties of ChameleonEC; and (ii) the experiments to understand the sensitivity of ChameleonEC.

How to start ChameleonEC: We run the prototype of ChameleonEC on Amazon EC2 to assess its performance.

```xml
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://Master:9000/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>Master,Slave2,Slave3,Slave4,Slave5,Slave6,Slave7,Slave8,Slave9,Slave10,
    Slave11,Slave12,Slave13,Slave14,Slave15,Slave16,Slave17,Slave18,Slave19,Slave20</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/root/zookeeper</value>
  </property>
</configuration>
```

Figure 2: Major configurations of `Hbase`.

Figure 1 shows the major configurations of our testbed. We use the following commands to start ChameleonEC:

We use the script `run-BD-Monitor.py` to start the bandwidth monitor.

```bash
#!/bin/bash
$ python3 run-BD-Monitor.py
```

We run the ECCoordinator on the Master to start the ChameleonEC coordinator.

```bash
#!/bin/bash
$ ./ECCoordinator
```

We run the ECHelper on the slaves to start the ChameleonEC helpers.

```bash
#!/bin/bash
$ ./ECHelper
```

We run the ECClient on the failed node to send a repair request.

```bash
#!/bin/bash
$ ./ECClient
```

We write a script `start.py` to start ECCoordinator and ECHelper conveniently and use `stop.py` to kill them.

```bash
#!/bin/bash
$ python3 start.py
$ python3 stop.py
```

Experiments with YCSB: We measure the impact of network bandwidth, chunk sizes, slice sizes, and erasure coding parameters under `YCSB`. `YCSB` relies on `Hadoop` and `Hbase`. We omit their installation process here. Figure 2 shows the major configurations of `Hbase`.

The following script starts `Hadoop` and `Hbase` to run `YCSB`:

```bash
#!/bin/bash
$ start-all.sh
$ start-hbase.sh
```

We use `YCSB WorkloadA` by default. Figure 3 shows the configurations of `YCSB WorkloadA`.

We use the following command to load data and run the `YCSB` server:

```
recordcount=200000
operationcount=200000
workload=site.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0.50
updateproportion=0.50
secanproportion=0
insertproportion=0

requestdistribution=zipfian

table=usertable
columnfamily=family

fieldlength=65536
fieldcount=1
```

Figure 3: Configurations of `YCSB` WorkloadA.

```bash
#!/bin/bash
$ ./root/ycsb-0.17.0/bin/ycsb load hbase20
-P /root/ycsb-0.17.0 /workloads/testworkload
-cp /root/hbase-2.3.6/conf/ -threads 2

$ ./root/ycsb-0.17.0/bin/ycsb run hbase20
-P /root/ycsb-0.17.0 /workloads/testworkload
-cp /root/hbase-2.3.6/conf/ -threads 2
```

Experiments with Memcached: We also employ `IBM Object Store`, `Memcached(Twitter)`, and `Facebook ETC workloads` to clarify how we assess the performance of ChameleonEC. The experiments with `Memcached` workloads are similar. We use `IBM Object Store` as an example. For every instance, we run the following commands to start `Memcached`:

```bash
#!/bin/bash
$ service Memcached stop
$ memcached -d install -m 8192 -u root -p
11211 -c 4096 -I 1M
```

We use `IBM-load` and `IBM-run` to load data and start `IBM Object Store` server.

```bash
#!/bin/bash
$ ./IBM-load
$ ./IBM-run
```

We write a script `IBM-run.py` to perform these operations. We also use `Facebook-run.py`, `Twitter-run.py` to run other `Memcached` servers.

```bash
#!/bin/bash
$ python3 IBM-run.py 0 (run IBM trace only)
$ python3 IBM-run.py 1 (run ChameleonEC only)
$ python3 IBM-run.py 2 (run IBM trace and ChameleonEC)
$ python3 IBM-run.py 3 (initialize and  load IBM trace
 data)
```